Twelfth USA/Europe Air Traffic Management Research and Development Seminar (ATM2017)

A Path Planning Algorithm to Enable Well-Clear Low Altitude UAS Operation Beyond Visual Line of Sight

Swee Balachandran
National Institute of Aerospace
Hampton, VA, USA

Anthony Narkawicz, César Muñoz, María Consiglio
NASA Langley Research Center
Hampton, VA, USA

***Abstract*—The availability of reliable and efficient algorithms to avoid obstacles, geofences, and other traffic is an essential functionality for safe autonomous operation of Unmanned Aircraft Systems (UAS) in low altitude airspace beyond visual line of sight. This paper presents a path planning algorithm that enables UAS to fly specified missions while accommodating real time traffic and geofence constraints. This planning algorithm integrates a rapidly exploring random tree planning technique with formally verified algorithms for maintaining well clear with respect to traffic aircraft and for detecting geofence conflicts. A simple heuristic that determines when to terminate tree expansion leads to low average computation times making this approach suitable for UAS with limited onboard computing power.**

Keywords – unmanned aircraft systems (UAS), detect and avoid (DAA), geofencing, path planning, autonomous systems

I. INTRODUCTION

Advances in embedded hardware and sensors have made it possible to build low cost Unmanned Aircraft Systems (UAS) that can be used for a wide range of applications such as package delivery, search and rescue, surveillance, infrastructure inspection, and data gathering. As more UAS take to the skies to perform these tasks, it is essential that they operate safely and within approved airspace constraints. For beyond visual line of sight (BVLOS) missions, where human intervention is unavailable, the capability to autonomously navigate and avoid obstacles, avoid keep-out geofences, stay within keep-in geofences, and detect and avoid other traffic aircraft is fundamental.

Mission planning when complete terrain and plan information are known a priori can be done efficiently offline. However, when traffic flight plans are unknown or when new constraints are imposed on the mission in real time, the original flight plan computed offline needs to be changed dynamically to accommodate the new constraints. This paper presents a planning algorithm that supports on-the-fly generation of local flight plans for solving real time geofencing and traffic constraints.

Large UAS, capable of flying in non-segregated airspace with manned aircraft, are required to be equipped with Detect and Avoid (DAA) systems [1] to aid remote pilots to comply with federal regulations to "see and avoid" other aircraft. DAA systems provide situational awareness in the form of traffic alerts and maneuver guidance intended to aid remote pilots in maintaining or regaining "well clear" separation with traffic aircraft. Small UAS (sUAS), operating at low altitudes (at or below 400 ft.) in uncontrolled airspace are also expected to maintain safe separation from other users of the airspace. While current safety regulations (FAA sUAS Rule (Title 14, Part 107)) [2] limit sUAS to remain within visual line-of-sight (VLOS) of the pilot-in-command at all times, it is expected that beyond visual line-of-fight (BVLOS) missions will have the greatest potential for economic impact. New regulatory guidelines are needed to help the implementation of these missions but in the meantime, it is clear that these tactical, contingency functions will have to be autonomously executed by the on-board systems.

Autonomous maneuver execution requires the implementation of new DAA functions that, for large UAS DAA systems were the responsibility of the pilot-in-command (PIC). This new functionality includes selecting a safe avoidance maneuver as well as a safe and efficient return to path maneuver. After execution of these maneuvers, the UAS must return to its original flight plan without human intervention. In low altitude airspace, in addition to avoiding other traffic, the UAS must also satisfy geofence constraints and avoid obstacles. Consequently, the integration of DAA algorithms with motion planning algorithms is essential for the safe autonomous BVLOS operation of UAS in low altitude airspace.

This paper presents the integration of a rapidly exploring random tree planning technique with DAIDALUS (Detect and Avoid Alerting Logic for Unmanned Systems) [3], a detect and avoid algorithm, and PolyCARP (Algorithms and Software for Computation with Polygons) [4], a geofence conflict detection algorithm. This integration yields a path planning algorithm that safely and efficiently computes alternate flight plans to resolve conflicts with other air traffic while satisfying geofence and obstacle constraints. A simple heuristic is used to determine when to stop exploring the search space which in turn significantly decreases the average computation time. The resulting algorithm is suitable for UAS with limited onboard computing resources. In cases where there is not enough time to compute a complete path to the goal, the algorithm provides a conflict free path to an intermediate goal state. Upon completion of the intermediate goal, the path planning process is repeated until the final goal is reached. The proposed

algorithm is applied to encounter scenarios involving multiple traffic and various geofence geometries. Computation time results for each of the encounter scenarios are presented.

The safety critical nature of several UAS applications mandates a rigorous understanding of the properties of the integrated approach. Consequently, the proposed planning algorithm uses algorithms from DAIDALUS and PolyCARP, which have correctness and safety properties that have been verified in the Prototype Verification System (PVS) [5]. A generic version of the path planning algorithm, which models random tree generation over an arbitrary node type, is formally specified and verified in PVS. The specific path planning algorithm presented in this paper is a special instance of this generic algorithm.

The algorithm presented in this work is integrated as part of a decision-making framework for UAS called ICAROUS (Integrated Configurable Algorithms for Reliable Operations of Unmanned Systems) [6][1]. This paper is organized as follows. Section II discusses related work. A brief overview of the key components and tools used in this work are reviewed in Section III. A detailed description of the path generation algorithm is presented in Section IV. Case study results are presented in Section V. The formalization of the proposed system in PVS is discussed in Section VI. Section VII provides discussions and future work. Finally, conclusions are provided in Section VIII.

II. RELATED WORK

A. Motion planning

The problem of finding a path from a given start configuration to a final configuration has been explored in depth in the literature [7, 8, 9, 10, 11, 12, 13, 14]. The various solution techniques in the literature can be roughly classified into two classes: graph based search methods and optimal control based methods. The graph based search methods discretize the search space into nodes. The nodes are connected based on some relevant criteria, e.g., reachability, to create a tree that spans the entire search space. This tree can then be searched for the shortest path using a graph based search algorithm, e.g., Dijkstra, A*, depth/breadth first, etc. In the optimal control based methods, the problem objective, e.g., shortest path, shortest time, minimum fuel, is typically encoded as a cost function. A parametric representation for the path or the control inputs to the system is chosen. Then, an optimization algorithm solves for the parameters that minimize the cost function subject to various constraints on the environment and dynamics.

The graph based methods suffer from the curse of dimensionality, i.e., the complexity of these algorithms typically increases exponentially as the dimensionality of the search space increases. The optimal control methods can be difficult to solve analytically. Hence, they are often based on numerical techniques. The presence of nonlinear constraints makes the optimization problem harder to solve. Furthermore, some minimization methods may yield locally optimal solutions. Certain optimal control techniques like Dynamic Programming also suffer from the curse of dimensionality.

Sampling based methods such as Probabilistic Road Maps (PRMs) [8] and Rapidly Exploring Random Trees (RRT) [10] were introduced to alleviate the complexity challenges of graph based search methods. These techniques have been used successfully in several applications to generate motion plans in real time. Some researchers have explored the application of sampling based planners to problems involving moving obstacles [11, 12].

B. Geofencing

Geofencing strategies for small UAS have been explored by some authors [15, 16]. The main focus so far has been on detecting geofence conflicts for remotely piloted UAS and preventing these vehicles from violating geofence constraints using predetermined maneuvers such as landing, return to home, or bouncing back from the fence. For fully autonomous operation, geofence conflict resolution must consider the mission requirements and dynamically compute new flight plans to avoid or go around geofences when possible.

C. Detect and Avoid

In the case of manned aircraft, several approaches have been proposed for ground-based and, more recently, airborne separation assurance systems (see, for example, [17]). Airborne technology for collision avoidance such as TCAS (Traffic Alerting and Collision Avoidance System) [18] has been successfully adopted by the commercial aviation industry. TCAS II, the second generation of TCAS, issues aural and visual alerts that direct pilots to maintain or increase vertical separation with intruders that are considered collision threats. The TCAS II system is based on an interrogation mechanism between transponders onboard the aircraft and a set of distance and time thresholds that determine the type of resolution advisories. A new generation of collision avoidance system, called ACAS X (Advance Collision Avoidance System), has been recently proposed [19]. In contrast to TCAS II, the alerting and advisory logic on ACAS is based on numerical look up tables optimized with respect to a probabilistic model of the airspace.

For UAS, the final report of the Federal Aviation Administration (FAA) Sense and Avoid (SAA) Workshop [20] defines the concept of *sense and avoid* as "the capability of a UAS to remain well clear from and avoid collisions with other airborne traffic." Based on this definition, the UAS Sense and Avoid Science and Research Panel (SARP) made a recommendation for a quantitative definition of UAS Well Clear that uses distance and time functions similar to those used in the TCAS II resolution advisory logic [21]. For large, remotely piloted UAS, the RTCA Special Committee 228 has developed minimum operational requirements for detect and

[1] https://github.com/nasa/ICAROUS

avoid that uses SARP's well clear definition [22]. DAIDALUS [3] is a NASA developed software that serves as a reference implementation of the detect and avoid concept provided by the RTCA SC-228 MOPS. This paper focuses on integrating detect and avoid systems, such as DAIDALUS, into a motion planning algorithm to facilitate low altitude autonomous UAS operations beyond visual line of sight.

III. BACKGROUND

This section gives an overview of the main components of the proposed path planning algorithms.

A. *Rapidly Exploring Random Trees*

Introduced by LaValle [10], a rapidly exploring random tree (RRT) is a data structure that can be used for path planning. Starting from an initial position, a tree is incrementally expanded towards randomly chosen samples in the search space. The tree expansion can be monitored at each iteration to ensure new nodes being added to the tree are consistent with any imposed constraints. If the random samples are chosen uniformly, each expansion step biases the tree towards the unexplored regions. Consequently, after several expansion steps, the tree spans the search space uniformly. Figure 1 illustrates a RRT generated from an initial position. The dark rectangles in the figure represent geofence obstacles. The tree data structure can then be utilized in conjunction with other graph based search algorithms to find a path from the root to any given node in the tree. The RRT data structure has many other properties that make it suitable for path planning. Interested readers are referred to [10] for more information.

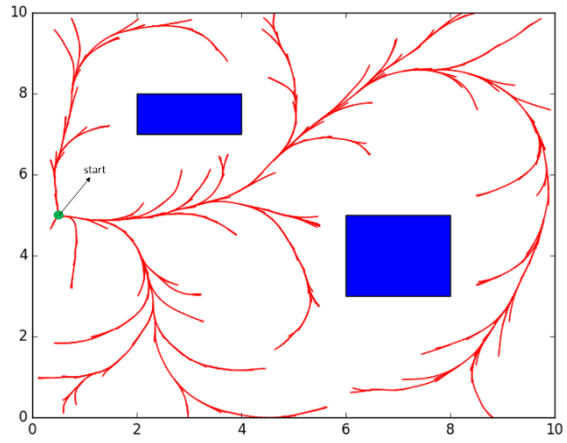

Figure 1: Exploration of search space using RRT

B. *DAIDALUS*

The detect and avoid functionality needed to implement the proposed path planning algorithm is provided by DAIDALUS [3]. DAIDALUS is a software library that serves as a reference implementation of the DAA concept described in the RTCA SC-228 Minimum Operational Performance Standards (MOPS) for Unmanned Aircraft Systems [1]. DAIDALUS source code is available in both C++ and Java under NASA's Open Source Agreement. The DAIDALUS software library consists of algorithms that predict well-clear violations between the ownship and traffic aircraft, and provide maneuver guidance in the form of range of maneuvers for the ownship to maintain or regain well-clear status with respect to traffic aircraft. These algorithms have been formally verified for logical correctness in the PVS verification system.

Figure 2 illustrates DAIDALUS functionality on a notional encounter. The solid area represents the well-clear volume that the aircraft needs to avoid. This volume is defined by time and distance thresholds and is generated assuming ownship performance limits. In this notional encounter, the current ownship trajectory is not predicted to be in conflict, but if the ownship maneuvers to the right in the range of maneuvers denoted by γ, the aircraft will eventually lose well clear. The range of maneuvers denoted by γ is called a *conflict band* [23]. A typical progression of conflict bands is illustrated in Figure 3 on a nominal encounter. At time t_0, aircraft are beyond threshold limits and therefore no bands are computed. At time t_1, the aircraft are within threshold limits and a peripheral conflict band is computed for the ownship. At time t_2, the intruder aircraft has maneuvered in the direction of the ownship and a conflict band appears in the current path of the ownship. At time t_3, the aircraft have lost well clear. In this case, DAIDALUS computes *recovery bands*, which is represented by the dashed green range of maneuvers. Recovery bands enable the ownship to regain well clear in a timely manner according to its performance limits.

DAIDALUS is a highly configurable system, but its default configuration was designed for large fixed wing UAS and high altitude operations. For the integration proposed in this paper, the well clear volume is defined as a cylinder of diameter 10m and height 10m. The performance limits of the ownship were also adapted to those of a small rotorcraft.

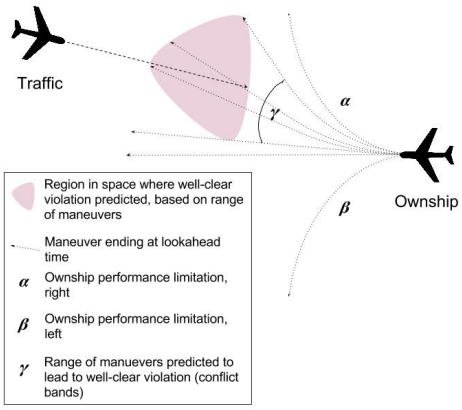

Figure 2: Maneuver guidance computation in DAIDALUS

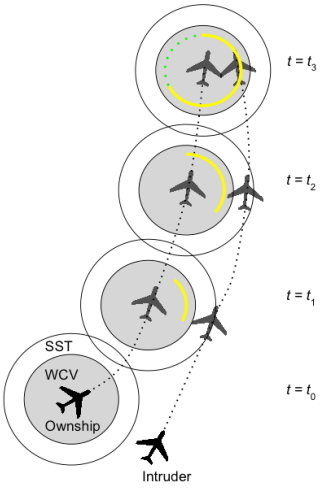

Figure 3: Progression of conflict bands

C. PolyCARP

The geofencing capability used by the proposed path planning algorithm is provided by PolyCARP [4]. PolyCARP is an open source software library developed by NASA that contains algorithms for detecting conflicts between a point moving at constant velocity and a (possibly moving) polygon. A core component of PolyCARP containment algorithm, with key functions that are formally verified, that determines if a point is definitively inside or definitively outside a polygon. This containment algorithm is partially based on a ray casting technique. Given a polygon region and an input position in a 2D plane, a ray is cast from the point outward to infinity. If it crosses an even number of edges of the polygon, it is outside; otherwise, it is inside. This is shown in Figure 4. In the containment algorithm, a parametric buffer distance is used to perturb the original polygon because ray casting may cause the ray to pass very close to some vertices, which can potentially allow floating point errors to produce an incorrect inside/outside result. The perturbation of the vertices away from the cast ray, prior to counting the number of crosses, mitigates this problem and the containment algorithm is correct assuming that the accumulated computation errors are less than the buffer and the point is less than the buffer distance away from an edge.

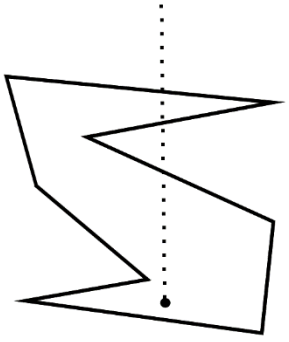

Figure 4: Ray casting

IV. PATH PLANNING

This section describes the integration of geofencing and detect and avoid capabilities into a path planning algorithm that efficiently computes a path from a given start configuration to a final configuration. The computed path is free from geofence and traffic conflicts.

A. Assumptions

The following assumptions are made to facilitate the development of the system:

- The ownship has a velocity controller. This enables the ownship to follow a commanded velocity.
- Complete information about all air traffic is available to the ownship.
- All intruder vehicles travel with a constant velocity.
- Intruder vehicles do not have malicious intent.
- All geofences are represented as polyhedra.
- All obstacles are encoded as keep-out geofences.

B. Problem Statement

Let $\mathbb{R}^3$ represent the 3-dimensional Euclidean space. Let $X_{start} \in \mathbb{R}^3$ represent the initial position of the ownship. Let $X_{end} \in \mathbb{R}^3$ represent the goal position for the ownship. Let $U \in \mathbb{R}^3$ represent a velocity command. Given X_{start} and X_{end}, the goal is to compute a sequence of velocity commands $U_0, \ldots U_n$ such that when applied to the ownship, yields a trajectory connecting X_{start} to X_{end} that (a) satisfies dynamics constraints of the ownship, (b) stays away from the well-clear violations, and (3) respects geofence constraints. The sequence of velocity control inputs is computed by randomly exploring the search space and incrementally building a tree data structure that spans both space and time.

C. Data structure

A tree is a collection of nodes that are connected by edges based on the availability of a feasible path between nodes. Each node contains information about the state of the ownship and intruder vehicles at a given time. The ownship state is represented as a vector of dimension n. The dimension n of the ownship state depends on how the dynamics of the vehicle are modeled. For example, the ownship state could be represented by a 6-tuple representing its 3-dimensional position and velocity vectors. Assuming each intruder's state is represented by a p-dimensional vector, q intruders can be represented by a $p \times q$ array. In addition to the ownship and traffic states, additional data such as the control input applied at the node, cost, and shortest distance to the goal node can also be stored within a node. Information about the parent and children edges are also stored within a node.

D. Tree Expansion

Let ΔT represent a fixed time interval allotted for the computation of a path. The root node in the tree is initialized with the initial position of the ownship and intruders projected in time by ΔT to account for the motion of the ownship during the computation process. At each iteration, a random configuration $X_r \in \mathbb{R}^3$ is generated from a uniform distribution. Then, a node in the tree is selected whose ownship position X_n is closest to X_r and such that there exists control inputs U that can move the ownship state from X_n closer to X_r within the interval $[t, t + dt]$. Here t represents the time at position X_n.

The above step of propagating the ownship state X_n towards X_r involves two stages. First, the control input U is computed. This computation largely depends on how one models the underlying dynamics of the ownship. Several approaches such as those described in [12] can be adopted. The control input is applied from t to $t + dt$ to generate a new state X_m. The computed input U is stored in the node corresponding to X_n. As the ownship position is propagated from t to $t + dt$ according to the modeled dynamics, the position of all the intruders are also propagated from t to $t + dt$.

E. Constraint Satisfaction

The segment from X_n to X_m is checked for geofence violation pointwise in time within the interval $[t, t + dt]$. To ensure that the new ownship position X_m is not in conflict with any intruders, well-clear guidance is computed using the functions that are part of DAIDALUS. Using the position and velocity of the ownship and intruders at time $t + dt$, DAIDALUS guidance maneuvers ensures that the state of the ownship at X_m is outside the well clear volume of any intruder. Furthermore, it is checked to ensure that the heading of the ownship at X_m lies outside any conflicting track angles that lead to a well clear violation. Finally, the change in heading from X_n to X_m is also checked that it does not cross any conflicting track angles.

The new position X_m is only reachable from X_n if the segment from X_n to X_m satisfies the above geofence and traffic conflicts. If X_m is reachable, then a new node with the state information of the ownship and intruders at $t + dt$ is created and added to the tree. If the above checks fail, X_m is discarded. Consequently, each node added to the tree is free from conflicts. With the addition of each new node to the tree, the expansion algorithm keeps track of the node N_c that is closest to the goal configuration.

With each iteration of the expansion step, the tree grows bigger leading to uniform exploration of the search space. This exploration phase is continued until a node is found that is within some neighborhood of the goal position. The sequence of inputs that must be applied from the root node to get to the goal node can be obtained by backtracking to the root node from the closest node to the goal N_c.

F. Heuristic-Based Termination

Often, depending on the geometry of the search space, it may not be necessary to explore the whole state space by growing the tree incrementally until a neighborhood of the goal position is reached. A simple heuristic to determine when to terminate tree expansion is to check if a direct path to the goal exists at a given node. More specifically, if one were to head directly towards the goal from a given node, and the path along this direction is free from traffic and geofence conflicts, then a direct path to the goal exists. This heuristic can be used on the nearest node used for tree expansion and/or on the closest node N_c.

If it is not possible to compute a complete path to the goal within the time allowed for computation ΔT, the algorithm returns the ownship position at the closest node N_c as the intermediate goal. The planning process can be reinitiated on reaching the intermediate goal.

G. Pseudo-Code

The pseudo-code for the algorithm is presented below. For convenience, a node at which the ownship's position is X is referred to as node X.

```
Initialize tree root with initial position and velocities of ownship and intruders.
Initialize closest node to goal N_c as an empty node.
For i in 1 to NumIterations
    Generate random sample X_r
    Find nearest node X_n in tree
    If direct path to goal node exists,
        Add goal node as child of X_n
    Else
        Find input U that move X_n closer to X_r
        Propagate X_n towards X_r with U to get X_m
        If segment from X_n to X_m satisfies traffic and geofence constraints, then
            Store input U in node X_n
            Add X_m as a child node of X_n
    Update closest node to goal N_c
    If goal reached, then
        Terminate
Back track from closest node to goal N_c to the root node, to obtain the path and control input sequence
```

H. Decision-Making Logic

The above algorithm assumes that all intruders travel with a constant velocity. If intruder vehicles change their trajectories, then the resolution planned obtained using the proposed algorithm may be obsolete if the new intruder trajectories lead to future conflicts. Consequently, it is necessary to be able to re-plan when new conflicts are imminent. Figure 5 illustrates a simple decision sequence that provides the capability to re-initiate the planning algorithm to ensure future conflicts are resolved appropriately.

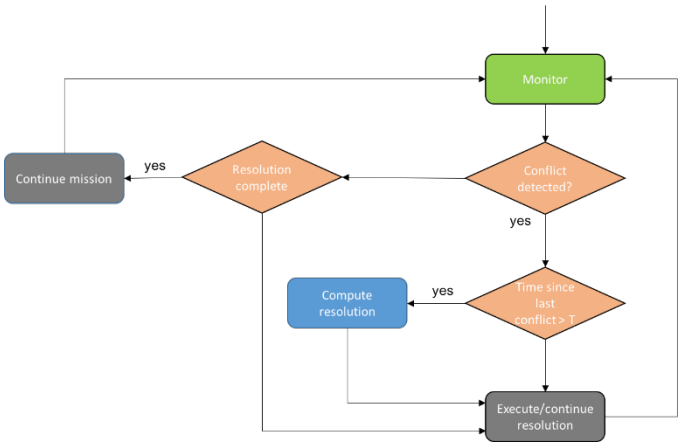

Figure 5: Decision making logic

V. CASE STUDY

The algorithm proposed in this work is agnostic to the vehicle type. However, for the purpose of illustration, a quadrotor vehicle is used. As mentioned earlier, it is assumed that the quadrotor has an onboard velocity controller and hence the quadrotor state is modeled as $X = [x, v_x, y, v_y, z, v_z]$. Here x, y and z represent the position of the quadrotor in a local North-East-Down (NED) coordinate frame. v_x, v_y and v_z represent the body velocities in the NED frame. The closed loop dynamics of the quadrotor is described by an ordinary differential equation of the form $\dot{X} = f(X, U)$ where $U = [u_x, u_y, u_z]$ represents the velocity control inputs.

$$\begin{aligned} \dot{x} &= v_x \\ \dot{v}_x &= -k_x(v_x - u_x) \\ \dot{y} &= v_y \\ \dot{v}_y &= -k_y(v_y - u_y) \\ \dot{z} &= v_z \\ \dot{v}_z &= -k_z(v_z - u_z) \end{aligned} \tag{1}$$

Here k_x, k_y, k_z are positive constants. For an expressive set of behaviors involving different modes of operation, the vehicle dynamics may also be represented as a hybrid system. For example, with a hybrid system model of a quadrotor dynamics, one can compute paths where the quadrotor can hover at a given position for a certain duration waiting for traffic related conflicts to be resolved.

A random sample is generated in the search space as follows:

$$\begin{aligned} x_r &= x_{min} + \mathcal{U}(0,1)\times(x_{max} - x_{min}) \\ y_r &= y_{min} + \mathcal{U}(0,1)\times(y_{max} - y_{min}) \\ z_r &= z_{min} + \mathcal{U}(0,1)\times(z_{max} - z_{min}) \end{aligned} \tag{2}$$

Here $\mathcal{U}(0,1)$ represents a uniform distribution in the interval $[0,1]$. The max, min values of x, y, z represents the bounds within which a random sample is generated.

The inputs that move the ownship position at the nearest node X_n towards the random sample X_r can be found using an optimization framework as discussed in [8]. For the simple quadrotor illustration, a simple feedback input of the form $U = k(X_r - X_n)$ is considered. Here k is a gain chosen to ensure $\|U\| \leq U_{max}$.

The ownship state X_n is propagated from time t to $t + dt$ using input U and the closed loop dynamics represented by Eqn (1). For the simplified closed loop dynamics considered in this work, it is possible to compute the solution analytically. However, more generally, one could use numerical integration techniques such as the Euler method or the Runge-Kutta method. Since each node also contains the position X_T and velocity V_T of the intruder vehicles at time t, we can also project the position of the intruder to time $t + dt$ using the linear dynamics of the form:

$$X_T(t + dt) = X_T(t) + V_T dt \tag{3}$$

This work assumes that the intruders move with a linear velocity. However, if the flight plans of all the intruders are known a priori, the above linear dynamics can be replaced with models representing the intruder trajectories.

As the positions of the ownship and intruders are incrementally projected from t to $t + dt$, it is ensured that each intermediate position between t and $t + dt$ is free from geofence conflicts pointwise in time. With the final positions and velocities of the ownship and intruder at $t + dt$ obtained, it is verified that there are no well clear violations and traffic conflicts as described in the previous section.

The application of the proposed algorithm combined with the decision-making logic in Figure 5 is illustrated in the Figures 6-9. The ownship (blue triangle) is travelling through as sequence of waypoints (indicated by the flight plan in black). The red triangles indicate intruder vehicles while the blue polygons indicate keep-out geofences. The conflict band output of DAIDALUS is illustrated around the ownship. The red bands indicate the ownship headings which can result in conflict with the intruders. The solid magenta curves indicate the paths explored by the algorithm. The dashed magenta line indicates the direct path to the next waypoint that is obtained by the early termination heuristic. Figures 6 and 8 illustrate two consecutive encounters. The first encounter is resolved by turning to the left and returning to the next waypoint after flying for a few seconds. This provides enough time for the first intruder to continue flying its course without entering a well clear violation with the ownship. The ownship encounters the second intruder before completing the previous resolution (see Figures 8-9). Due to the decision-making logic discussed above, a new resolution is generated to avoid the second intruder. The computed resolutions ensure that the keep-out geofence constraints are satisfied at all times.

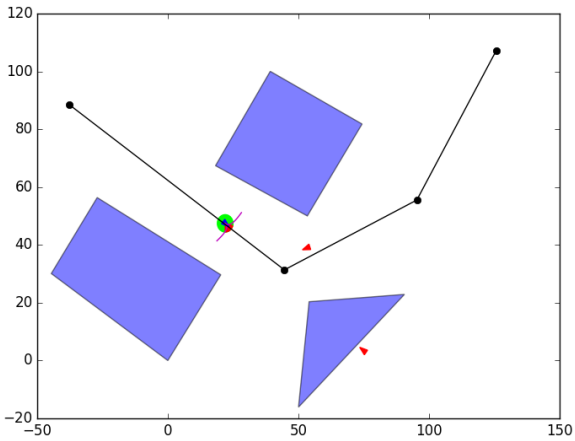

Figure 6: Traffic encounter 1

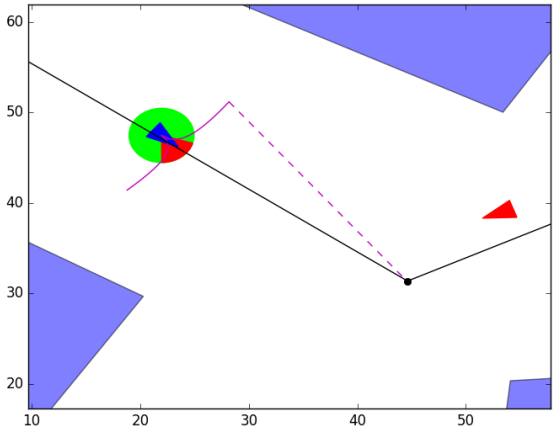

Figure 7: Close up of encounter 1

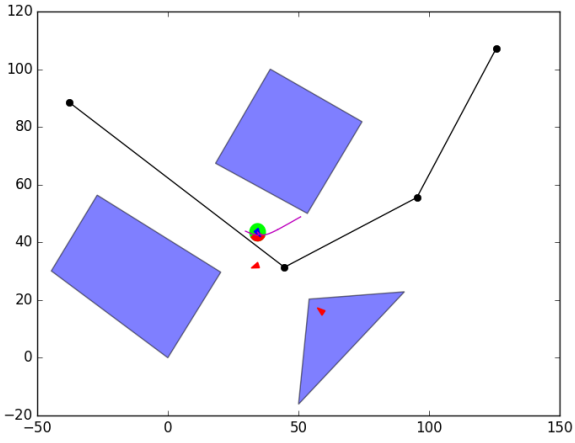

Figure 8: Traffic encounter 2

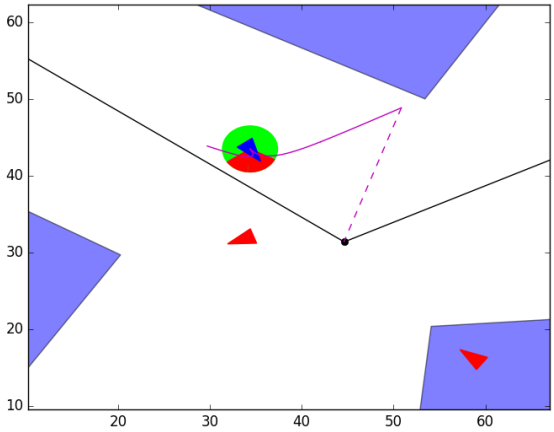

Figure 9: Close up of encounter 2

The computation times for the proposed algorithm were evaluated on two different hardware platforms suitable for embedded applications. Results are summarized in Tables 1 and 2.

Table 1: Computation time results on Beagle Bone Black [24]

	Iterations used	Nodes explored	Time taken (s)
Encounter 1	5	5	1.5572
Encounter 2	7	6	1.5802
Encounter 3	16	13	2.6758

Table 2: Computation time results on Jetson TK1 [25]

	Iterations used	Nodes explored	Time taken (s)
Encounter 1	7	5	0.1324
Encounter 2	19	14	0.2822
Encounter 3	10	7	0.1671

VI. Formalization and Verification

As noted in the Introduction, the safety critical nature of UAS algorithms mandates a rigorous understanding of the properties of the integrated approach. Consequently, many of the core algorithms in DAIDALUS, PolyCARP, and ICAROUS have been formally verified in the Prototype Verification System (PVS) [5]. PVS is an *interactive theorem prover* consisting of a specification language and a prover language. Algorithms are first specified in the specification language; whose semantics are similar to a functional programming language. However, unlike a programming language, this specification language allows correctness properties, in the form of *theorems,* to be stated about these algorithms. The prover language is then used to *prove* that the algorithms satisfy these theorems.

In the PVS development, priority is given to verifying algorithms that have potential safety implications. The previously verified algorithms in DAIDALUS therefore include the conflict detection and bands algorithms. In PolyCARP, verified algorithms include sub-functions of the ray casting

algorithm, including the function that determines whether a ray crosses an edge, the function that determines if the input point is within a buffer distance of an edge, and the function that determines if a trajectory will intersect a polygon.

The Runge-Kutta approximation of the closed loop dynamics of the quadrotor, presented in Section III, have been formalized in PVS, as well as a function that checks the resulting path for well-clear (using DAIDALUS) and collisions with obstacle polygons (using PolyCARP). These algorithms are called as functions of the path planning algorithm presented in this paper. A generic version of the path planning algorithm has been formalized and verified correct in the PVS theorem prover. It allows random growth of trees to be modeled, where the type of the node objects is arbitrary. There are generic correctness properties that are checked before adding each new node to the growth tree, and assuming that these correctness properties satisfy certain soundness conditions, the final tree is proved to satisfy a general safety property. An implementation of the specific path planning algorithm presented in this paper is underway where the node objects include multiple pieces of information about the state of the aircraft. In this implementation, the general safety property will reduce to a property that all paths through the generated tree are well-clear and avoid obstacle polygons.

VII. DISCUSSION

The selection of a suitable goal location for plan generation is mission dependent and must be done in real time. A simple approach would be to select the next conflict free waypoint in the flight plan or an intermediate point in space on the current flight plan's leg as the goal node.

The available time ΔT for computing a solution is also problem dependent. Based on the availability of sufficient ΔT, one can ignore the early termination strategy discussed in this paper and focus on the tree expansion as desired. A graph based search algorithm may also be used on the tree to determine an optimal path. Certain scenarios may require using the early termination heuristic to trade off optimality for a cheap conflict free solution. Determining this tradeoff in real-time is a topic of interest for future work. Future research directions will also look into construing efficient exploration heuristics to select new samples.

VIII. CONCLUSIONS

This paper presented a local path planning algorithm that integrates a rapidly exploring random tree based search algorithm with resolution and geofence conflict detection algorithms that have formally verified components. Termination of the tree expansion based on the availability of a direct path to the goal position from a node on the tree leads to quicker computation times. Initial software and hardware in the loop tests show that the resulting algorithm is suitable for UAS with low computation power given its capability to compute in real-time motion plans that satisfy traffic and geofence constraints.

REFERENCES

[1] RTCA, "Detect and avoid (DAA) minimum operational performance standards (MOPS) for verification and validation (DRAFT)," Paper No. 261-15/PMC-1400, RTCA Inc, Washington, D.C, 2015.

[2] U. Government, "Electronic Code of Federal Regulations," [Online]. Available: www.ecfr.gov. [Accessed January 2017].

[3] C. Muñoz, A. Narkawicz, G. Hagen, J. Upchurch, A. Dutle, M. Consiglio and J. Chamberlain, "DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems," in *Proceedings of the 34th Digital Avionics Systems Conference (DASC 2015)*, Prague, Czech Republic, 2015.

[4] A. Narkawicz and G. Hagen, "Algorithms for collision detection between a point and a moving polygon, with applications to aircraft weather avoidance," in *14th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference, AIAA-2014-2859*, Kissimmee, Florida, 2015.

[5] S. Owre, J. Rushby and N. Shankar, "PVS: A Prototype Verification System," in *in Proceeding of the 11th International Conference on Automated Deduction (CADE), D.Kapur, Ed,. Lecture Notes in Artifical Intelligence, vol, 607, Springer, pp 748-752. 1992.*, 1992.

[6] M. Consiglio, C. Muñoz, G. Hagen, A. Narkawicz and S. Balachandran, "ICAROUS Integrated Configurable Algorithms for Reliable Operations of Unmanned Systems," in *Proceedings of the 35th Digital Avionics Systems Conference (DASC 2016)* , Sacremento, CA, 2016.

[7] J.-P. Laumond, "Robot Motion Planning and Control," in *Lectures Notes in Control and Information Sciences*, vol. 229, Springer, 1998.

[8] L. Kavraki and J. Latombe, "Probabilistic Roadmaps for Robot Path Planning," in *Practical Motion Planning in Robotics: Current Approaches and Future DirectionsPractical Motion Planning in Robotics: Current Approaches and Future Directions*, K. Gupta and A. del Pobil, Eds., John Wiley, 1998, pp. 33-53.

[9] Y. Hwang and N. Ahuja, "A Potential Field Approach to Path Planning," *IEEE Transactions of Robotics and Automation,* vol. 8, no. 1, pp. 23-32, February 1992.

[10] S. LaValle and J. Kuffner, "Randomized Kinodynamic Planning," in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, 1999.

[11] D. Hsu, J. C. Kindel, J.-P. Latombe and S. Rock, "Randomized Kinodynamic Motion Planning with Moving Obstacles," in *Proc. Workshop on Algorithmic Foundations of Robotics (WAFR'00)*, Hanover, NH, 2000.

[12] E. Frazzoli, A. D. Munther and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics,* vol. 25, no. 1, pp. 116-129, 2002.

[13] L. Dubins, "On Curves of Minimal Length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics,* vol. 79, pp. 497-516, 1957.

[14] D. Bertsekas, Dynamic Programming and Optimal Control, Belmont, MA: Athena Scientific.

[15] M. Stevens and E. Atkins, "Multi-Mode Guidance for an Independent Miltocopter Geofecing System," in *16th AIAA Aviation Technology, Integration, and Operations Conference*., 2016.

[16] E. Dill, S. Young and K. Hayhurst, "SAFEGUARD: An assured safety net technology for UAS," in *Digital Avionics Systems Conference (DASC)*.

[17] J. Kuchar and L. Yang, "A review of conflict detection and resolution modeling methods," in *IEEE Transactions on Intelligent Transportation System, 1(4):179-189* , December, 2000.

[18] T. Williamson and A. S. Ned, "Development and operation of the traffic alert and collision avoidance system (TCAS)," in *Proceedings of the IEEE 77.11*, 1989.

[19] M. J. Kochenderfer, E. H. Jessica and P. C. James, "Next generation airborne collision avoidance system," *Lincoln Laboratory Journal,* vol. 19, no. 1, pp. 17-33, 2012.

[20] "Final Report, FAA Sponsored Sense and Avoid (SAA) for Unmanned Aircraft Systems (UAS) Workshop," 2009.

[21] C. Muñoz, A. Narkawicz and J. Chamberlain, "A TCAS-II resolution advisory detection algorithm," in *Proceedings of the 2013 AIAA Guidance, Navigation, and Control Conference and Exhibit, no. AIAA-2013-4622*, Boston, Massachusetts, 2013.

[22] S. Cook, D. Brooks, R. Cole, D. Hackenberg and V. Raska, "Defining Well Clear for Unmanned Aircraft Systems," in *Proceedings of the 2015 AIAA Infotech @ Aerospace Conference, Kissimmee*, Florida, 2015.

[23] J. Hoekstra, R. Ruigrok, R. van Gent, J. Visser, B. Gijsbers, M. Valenti, W. Heesbeen, B. Hilburn, J. Groeneweg and F. Bussink, "Overview of NLR free flight project 1997-1999, Tech Rep. NLR-CR-2000-227," National Aerospace Laboratory (NLR), 2000.

[24] BeagleBoard.org Foundation, "Beagleboard," [Online]. Available: http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf. [Accessed 20 April 2017].

[25] Nvidia, "Nvidia Emedded Systems," Nvidia, [Online]. Available: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html. [Accessed 20 April 2017].